UCSD p-System 最新版の設計思想と言語体系

栗原好昭 （日本ビジネスオートメーション株式会社）

はじめに

現在多くの種類のマイクロプロセッサを使い、多様なスモールコンピュータが作られ、その高機能化と低価格化により年々その市場を拡大している。これと並行してプログラムへの需要が高まり、機種選定の条件として、「ソフトウェアが豊富」であることが重要な要素となってきた。もし多くの機種で共通に使えるソフトウェアが出来れば、供給者は一つのソフトウェアの開発により大きな市場が確保出来、利用者は新機種を導入した時に、それまでのソフトウェアの蓄積をむだにしなくてもよい。

UCSD p-System は、プロセッサや機種に依存する部分を少なくし且つ、モジュール化することにより、多くの機種への移植性を向上させ、ソフトウェアの互換性を実現している。

1 UCSD p-System とは

UCSD p-System はカリフォルニア大学サンディエゴ(UCSD)校で開発されたUCSD Pascalの最近版である。

UCSD Pascalは同校の教授であったDr. Kenneth L. Bowlesが中心となりPDP-11上に実現した。その後図1の系譜をたどり現在バージョンIV.1が米ソフテックマイクロシステムズ社により販売されている。

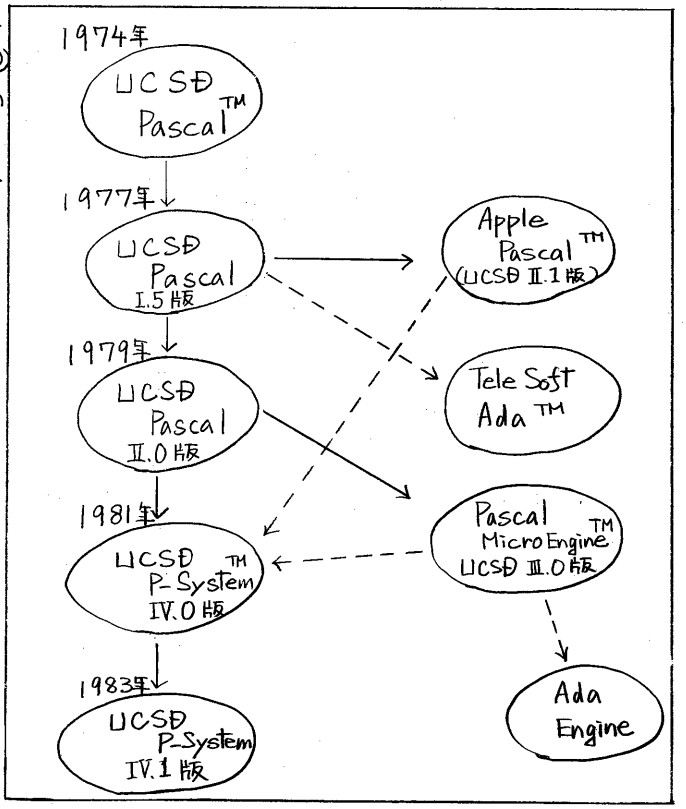

図1 UCSD p-System の系譜

2 中間コード(Pコード)方式

UCSD p-Systemの移植性のキーポイントは、中間コードによるコンパイラ・インタプリタ方式にある。この方式は高級言語のプログラムをコンパイラでプロセッサにより異なる機械語にせず、仮想16ビットプロセッサ(Pマシン)の機械語(Pコード)を作り、これを各プロセッサの機械語で作られたプログラム(Pコードインタプリタ又はPマシンエミュレタ)により翻訳実行する方式である。p-SystemやPコードの"P"は仮想(PSEUDO)の頭文字を取っている。p-Systemのオペレーティングシステムを含むほとんどのプログラムはPascal言語で作られており、コンパイラによりPコードで出力されている。このため、Pコードインタプリタが用意されていれば簡単に移植が可能である。現在表1に示すプロセッサには、インタプリタが用意されている。

中間コードの実行は、インタプリタ方式だけでなく、マイクロプログラムにより直接Pコードを実行するものと、Pコードをネーティブコードジェネレタにより各プロセッサの機械語に変換して実行するものもある。

8ビット系	16ビット系
8080/8085	8086/8088
Z80	9900/99000
6502	68000
6800*	PDP/LSI-11
6809*	Z8001

*印のCPUは現在のところII.0版のみ

表1. インタプリタの用意されているCPU

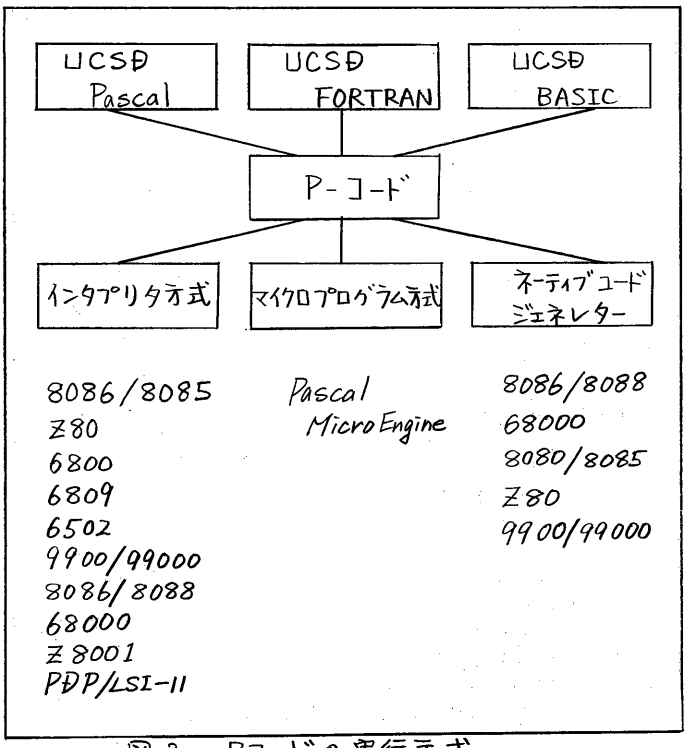

図2 Pコードの実行方式

3 機能分割

中間コード方式によりCPUへの移植性が向上したが、機能の分割により各機種への移植性が向上している。P-Systemでは図3に示すアーキテクチャによりCPUや機種毎に異なる部分を分離し、新たな機種に移植する作業が簡単になっている。

P-Systemは大きく3つの部分に分かれている。1つは全機種に共通なP-コードで作られた部分でオペレーティングシステム、コンパイラ、エディタ、アプリケーション等がこれに含まれる。2つ目は、P-コードを実行するためのCPU毎に用意されているインタプリタである。これによりP-コードで作られた部分は、CPUから分離されている。3つ目は各機種の周辺装置により異なるBIOS部でインタプリタを周辺装置から分離している。また、インタプリタとオペレーティングシステムは、さらに機能の分割が行なわれている。例えばオペレーティングシステムは、10数個の独立したモジュールにより構成されている。それぞれのモジュールは、ファイル入出力、画面制御、メモリ管理、タスク管理、組み込み関数等システムの各機能を受け持っている。

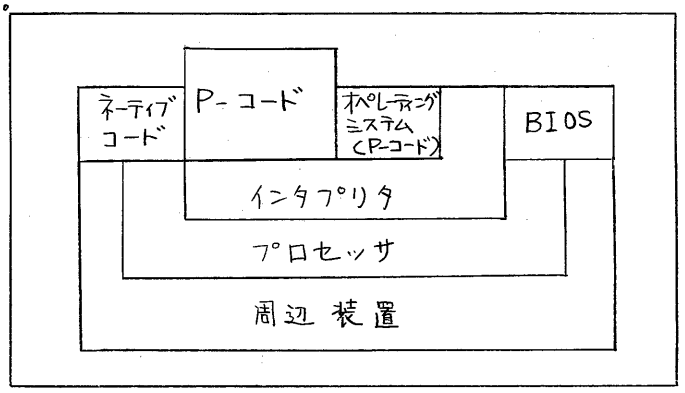

図3 P-Systemアーキテクチャ概略

P-System を移植する場合幾つかのステップがある。最も簡単な場合は、ターミナルが異なる場合で、ターミナル情報データファイルを作り直し、オペレーティングシステムへ10ステップ程度のプログラムを組み込むだけである。この作業は数時間で出来る。周辺装置が異なる場合は各装置用の入出力ルーチン(BIOS)を作成する必要がある。この作業は数日～数週間程度で行なえる。最も大変なのは、インタプリタが用意されていないCPUの場合であるが、それでも数ヶ月～10数ヶ月、人で充分である。しかし表1に示した様に現在使われているほとんどのマイクロプロセッサはインタプリタが用意されている。

4 分割コンパイルとライブラリ機能

プログラム開発を行なう場合、幾つかのプログラムで共通に使用するルーチンをライブラリとして管理する機能があると便利である。また大きなプログラムの開発時、機能別に分割して作成、テストをすることが必要となる。この2つの目的をP-Systemではユニットにより達成している。P-System を構成する各ソフトウエアもユニットによりシステムの移植性、メンテナンス性が向上している。

ユニットは2つの部分に分かれている。1つは、インタフェース部であり、メインとなるプログラム(ホスト)から使えるデータ構造とルーチンを定義する。この部分はユニットをコンパイルする時にテキストのままコード中に出力される

。そしてユニット宣言(USES)によりホストプログラムの一部としてコンパイルされ、名前や型の引用とチェックが行なわれる。もう1つは、インプリメンテーション部でそのユニットだけで使用するデータやルーチンと、インタフェース部で宣言されたルーチンの本体を拡納する。

例えば、P-Systemのグラフィックパッケージ"タートルグラフィックス"は各種のマイクロコンピュータで使われている。タートルグラフィックスのインタフェース部はどれも同じであるが、インプリメンテーション部は各マイクロコンピュータのハードウェア構成により異なっている。このため、ある機種用に作成したタートルグラフィックを使ったホストプログラムは、タートルグラフィックの使える別の機種でもそのまま利用可能である。

```
Program Example ;          | UNIT GRAF-UNIT;
USES GRAF-UNIT ;           | INTERFACE
VAR                        |  (
  (                        | PROCEDURE LINE;
                           |  (
BEGIN                      | IMPIEMENTATION
  (                        |  (
 LINE;                     | PROCEDURE LINE;
  )                        | BEGIN ~ END;
END.                       |  )
                           | END.
```

図4 ユニットとユニットを使用するプログラム例

5 セグメントルーチン

スモールコンピュータの限られたメモリで大きなプログラムを実行可能とするためにセグメントルーチンが用意されている。セグメントルーチンは、コードファイル中に独立したコードセグメントを作成する。このコードセグメントは、実行時にメモリが不足するとオーバレイが行なわれる。セグメントルーチンは手続き(関数も含む)宣言の前に"Segment"と付けることにより行なう。コードセグメントを含むプログラムは、ロード時にメモリに入る限りのコードセグメントが読み込まれる。もしメモリにロードされていないセグメントが呼び出されると、現在使われていないセグメントを消し(スワップアウト)必要なメモリを確保してロード後実行する。この操作はオペレーティングシステムにより自動的に行なわれる。しかしオーバレイが発生すると、ディスクの読み込みにより時間の遅れが生じる。これによりプログラムの機能がそこなわれないようにするため、プログラムで特定のセグメントをメモリ上にロックすることも可能である。

```
PROGRAM A;
  (
Segment Procedure X;   ┐ セグメント
Begin                  │  ルーチン
End;                   ┘
  (
Begin                  ┐ 主セグメント
End.                   ┘ (オーバレイされない)
```

図5 セグメントルーチンの記述

6 メモリ管理

セグメントオーバレイや動的変数割り付けによるメモリ要求を効率良く処理するために、オペレーティングシステムはメモリを幾つかの部分に分けて動的に管理している。メモリ管理の方法には図6に示す2種類がある。単一のメモリ空間を分割して管理する方法(図6の①)は、主に8ビット系CPUで使われる。スタックはローカル変数・演算スタックとして使われ、実行にともない低いアドレスへ伸びる。ヒープは動的変数の割り当てに使用し、高いアドレスへ伸びる。この間がコードプールとして使われ、プログラムがセグメントを単位として入る。もしスタックの下側がコードプールと重なりそうになると、コードプールが下に移動される。逆にヒープの上側がコードプールと重なりそうになると、コードプールが上へ移動される。コードプールの移動が出来ない場合、コードプール内の不要なセグメントをスワップアウトしてメモリを確保する。

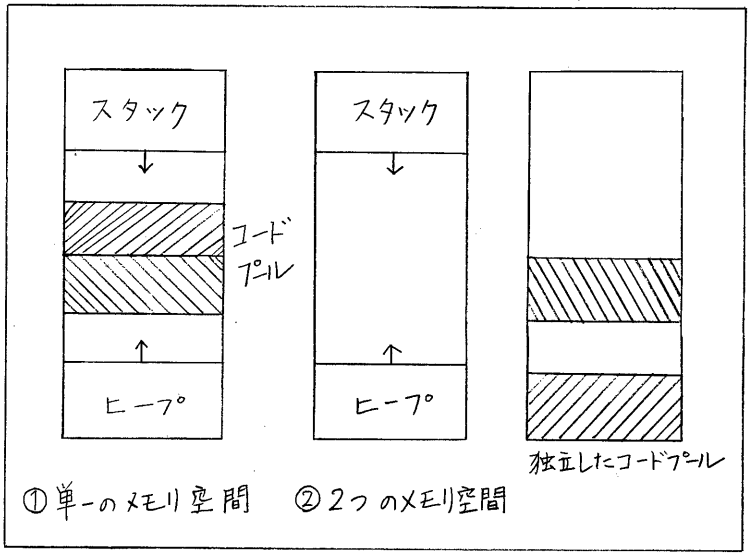

図6 p-Systemの2つのメモリモデル

2番目のメモリ管理の方法は、より多くのメモリを持つ16ビット系CPUで使われる。スタックとヒープは①の場合と同じであるが、コードプールは独立したエリアに置かれる。この2つのメモリ管理方式により適切にセグメント化されたプログラムを64Kバイトのメモリ空間の8ビットCPU上でも、128Kバイトのメモリ空間を持つ16ビットCPUでも実行することが可能となった。

7 ネーティブコード

p-Systemでは、少ないメモリ容量しか持たない。スモールコンピュータでも充分な大きさのプログラムが実行出来る様に最適化されたPコードにより実行が行なわれる。このPコード方式により移植性においても最適なものが得られた。しかし特定CPU上でPコードとそのCPUの機械語(ネーティブコード)で作られたプログラムの実行速度を比較すると、ネーティブコードの方が速い。またCPUに依存する処

理を記述するためにネーティブコードのルーチンと高級言語をリンクして使用する必要も生じる。前者の問題を解決するために用意されたのがネーティブコードジェネレタ(NCG)であり、後者の問題のためにアダプタブルアセンブラが用意されている。

ネーティブコードジェネレタはCPU毎に独立したプログラムとして作られている。プログラムの一部分をネーティブコードにする場合、ソースプログラムに特別な印を付けて通常のコンパイルを行なう。出力されたオブジェクトプログラムは全てPコードで、このままで通常の実行も可能である。このオブジェクトプログラムをネーティブコードジェネレタに入力すると、Pコードとネーティブコードが混在したオブジェクトプログラムを出力する。多くのプログラムにおいて、全体の数分の1のコードの実行に大部分の時間を使っている。この多くの時間を使う部分だけをネーティブコードとすることで、実行時間とコードサイズの最適化が行なえる。幾つかのCPUでネーティブコードとPコードの実行時間とコードサイズを比較した結果を表2と3に示す。

表2 Pコードとネーティブコードの速度比較

	Pコード	ネーティブコード	比
9900	121	10	12.1
Z80	73	14	5.2
8086	84	10	8.4

表3 Pコードとネーティブコードの大きさの比

	大きさ	比
Pコード	71	1
9900	166	2.3
Z80	234	3.3
8086	166	2.3

ネーティブコードを含むプログラムは、特定CPU以外では実行出来ない。このため、流通ソフトウエアで問題となるかもしれない。しかしネーティブコードジェネレタを通す前のプログラム(ネーティブコードにする部分に印を付けた)を供給すれば利用者が、各自のCPU用のネーティブコードジェネレタにより変換することができるし、遅いことをがまんすればPコードのまま実行も出来る。

ネーティブコードルーチンは、p-Systemに含まれている各CPU用のアダプタブルアセンブラにより作られる。このマクロアセンブラは、高級言語から呼び出す手続き(又は関数)として作られたアセンブリ言語を入力として、コンパイラが出力するオブジェクトと同一型式のリロケータブルなコードを出力する。このコードは高級言語のホストプログラムと専用のリンカによりリンクして実行する。アダプタブルアセンブラは、強力な1パスアセンブラで条件アセンブル、マクロ、アセンブリルーチン相互又は高級言語のグローバル定数・変数の参照定義が可能である。またp-Systemの管理外で使用するアセンブリルーチンの開発に使用するためのリロケータも用意されている。

8 マルチタスキングとイベント

p-Systemではシングルユーザ・マルチタスクをサポートしている。タスクは手続きと同じ型式で作られ"PROCEDURE"の変りに"PROCESS"を使う。タスクの制御はセマフォにより行なわれる。p-Systemのマルチタスキングは全ての機種に同一の機能を提供するため、時分割併行処理や入出力待ちによるタスク切り換えは

行なっていない。タスクの切り換えは、タスクの開始(START)、終了(プロセスの最後まで実行された)、同期(SIGNAL, WAIT)及び外部要因(EVENT)によって起る。タスクは、実行中、実行待ち、セマフォ待ちの3つの状態がある。実行待ちとは、実行可能な状態ではあるが、優先度が低いため待っている状態である。セマフォ待ちとは、WAIT 命令により当該セマフォにSIGNALまたはEVENT が発生するのを待っている状態である。

9 UCSD p-System の言語体系

p-System上で数多くの言語が使用可能である。p-Systemの開発・供給を行なっているSMS(ソフテックマイクロシステムズ)社からは、UCSD Pascal、UCSD FORTRAN、UCSD BASICが供給されている。この3種類の言語は相互間のリンクが可能である。これ以外の言語としてModula II、Ada、Occam等も使用可能である。ここではUCSD Pascal、FORTRAN、BASICに付いて概要を紹介する。

UCSD Pascal言語

Pascal言語はチューリッヒ工科大学のN. Wirth教授が設計した構造化プログラミング言語である。UCSD Pascalは、このPascalを基本にスモールコンピュータ上で各種のプログラム作成が可能となるよう機能の拡張が行なわれている。ここでは追加された機能に絞って説明する。

(1)文字例(ストリング)型と文字列処理用手続き
文字列を扱うために専用の型が追加されている。これは文字配列と異なり有効文字数を含んでいる。文字列は代入、比較、入出力と専用の手続きによる連結、有効文字数の取り出し、移動、部分取り出し、検索、数値からの変換が行なえる。

(2)入出力処理の強化
スモールコンピュータは会話型処理が多い。このため会話型入出力のためにインタラクティブ型が用意されている。標準入出力(INPUTとOUTPUT)もこの型となっている。また乱アクセスファイルを扱うためのSEEK手続きや低レベル入出力用にユニット入出力、ブロック入出力が用意されている。

(3)多精度整数
UCSD p-System の標準整数は2バイトで表現出来る範囲は±32767である。事務計算の分野では、これでは不充分である。このため、10進36桁までの精度が指定できるロングインテジャ型が用意されている。ロングインテジャ型は、四則演算、比較、入出力が標準の整数と同様に行なえる。

(4)セグメントルーチン・ユニット・マルチタスキング
これらの機能を扱うための型や文、手続きが用意されている。

UCSD FORTRAN

UCSD FORTRANはANSI FORTRAN-77に準拠し、多少機能の拡張と制限が行なわれている。UCSD FORTRANコンパイラはPコードを出力し、インタプリタにより実行する。実行にはFORTRANランタイムライブラリを必要とする。

UCSD FORTRANは、ANSI FORTRAN-77の2つの機能に制限がある。1つは、パラメータに手続名(副プログラムヌは関数副プログラム)を渡せないことである。もう1つは、整数と実数が同じ大きさではないことである。p-Systemにおける整数は2バイトであり、実数は4ヌは8バイトである。

UCSD FORTRANの拡張された機能は、コンパイラディレクティブ、会話向き出力編集、EOF関数である。

コンパイラディレクティブには、ソーステキストのインクルード、ユニットの宣言と使用、クロスリファレンス、外部ルーチン(アセンブラ言語ルーチン)の使用がある。

UCSD BASIC

UCSD BASICはコンパイラ型のBASICで、エディタで作成したソースプログラムはコンパイラでPコードに出力され、実行が行なわれる。UCSD BASICは、ANSIミニマルBASICを基にし機能の拡張が行なわれている。以下にUCSD BASICの特徴的な機能について説明する。

(1)行番号が不要

UCSD BASICは、行番号は必要な部分だけでよく、昇順である必要もない。これはp-Systemに用意されたスクリーンエディタによる入力と、コンパイラ方式だからである。またプログラムの段付けも自由である。

(2)仮想配列

これは、配列をメモリ上ではなく、ディスクに作成する方法で、大きな配列を扱う場合有用である。

(3)サブルーチンへのパラメータ渡し

サブルーチンの呼び出しは、GOSUBによる方法以外にCALLによる方法が用意されている。これは、パラメータの受け渡しが可能で、特にPascal、FORTRANルーチンを使う場合に必要となる。

10 UCSD p-Systemの問題点と今後

UCSD p-Systemの問題点の多くは、急速なハードウェアの低価格化と高機能化によるものである。この中で特に問題となるものは、メモリ容量の増大である。p-Systemは、現在のところ128Kバイトまでのメモリ管理能力しかない。8086、68000等のプロセッサを使用したシステムでは、これ以上のメモリを実装したシステムも多くある。この制限はほとんどのプログラムでは問題とならず、多次元、多要素の配列を使う場合位であるが、使われないメモリの存在自体に不満を持つユーザも多い。また全てのプロセッサで共通のオペレーティングシステムを使うため、16ビットプロセッサでは一部の機能が生かされないこともありうる。これらの問題点は次の版以降で解決されるであろう。

UCSD p-Systemは、今後もソフトウェアの互換性、移植性を保ちつつ、さらに機能の向上が行なわれるであろう。現在、作業中もしくは予定されている新たな機能を紹介する。

(1)ローカルエリアネットワーク(LAN)

p-Systemのオブジェクトレベルでの互換性は、ローカルエリアネットワークでさらに生かされるであろう。現在すでにコルブス社のオムニネットで実用化されているものもある。今後さらに一般的なネットワークサポートを行なう予定である。

(2) アドバンスド ファイル システム

現在 p-Systemでは2段階までのファイル構造をサポートしている。現在計画中のアドバンスドファイルシステムでは、UNIXと同様な木構造のファイルシステムとなっている。また、KEY付きシーケンシャルファイルのサポートも行なう。

(3) マルチユーザサポート

p-System のマルチユーザサポートは、幾つかの方法が試みられている。
UNIX オペレーティングシステム上の1つのプロセスとしてp-Systemを動かす方法は、p-System自体の大幅な改造を必要とせず最も簡単な方法で、PDP-11, UX-300等で実現されている。
インタプリタの改造により、一つのインタプリタで複数のオペレーティングシステムを動かす方法は、ALTOSで行なわれた。単一のインタプリタ、オペレーティングシステムでマルチユーザを実現する事はまだ行なわれていないが、最終的にはこの方向になるであろう。

おわりに

独特のアーキテクチャにより、ソフトウェアの互換性を実現したUCSD p-Systemに付いて説明した。この様なソフトウエアの普及により、ソフトウエアの流通がより活発になることであろう。

参考文献

(1) Mark Overgard, SofTech Microsystems
The UCSD p-System : A Portable Environment for Development and Execution of Parsonal Computer Application Software (1982)

(2) SofTech Microsystems, 日本ビジネスオートメーション訳
UCSD p-System ユーザーズマニュアル 上・下 及び サプリメント (1982)

(3) SofTech Microsystems
FORTRAN User reference Manual (1981)

(4) SofTech Microsystems
BASIC User reference Manual (1981)

(5) Stephen C. Koehler, William P. Franks
Native-Code Generation in an Emulation-Based Environment (1982)